ENCRYPTED DATABASE DESIGN: SPECIALIZED APPROACHES

Neal R. Wagner[1]
Drexel University

Paul S. Putter[2]
The Pennsylvania State University

Marianne R. Cain[1]
Drexel University

Abstract

This paper presents two specialized approaches to the design of a database incorporating cryptography in a fundamental way. Most processing is carried out on the encrypted version of the database at an insecure central site. Query initiation and final query interpretation occur at a secure local workstation. The first approach implements a text file with searches based on keywords. The second approach uses subfields and homophonic representations to create a secure database with fairly broad capabilities.

1. Introduction

There is an obvious role for encryption in the transmission or storage of any sensitive data, in particular of the data in a database. The need for a decryption key provides an extra layer of security beyond simple physical security. Encryption can also provide simple implementations of specialized functions, such as multilevel security via different encryption keys or secure filters via signatures. (See Section 2.)

One could just separately encrypt each record of a database and decrypt as needed. This involves a great deal of encryption / decryption and complicates or eliminates efficient lookup schemes, join operations, associative memory schemes, and other features of a modern database management system (DBMS) that we have come to expect.

A true encrypted database should have a DBMS that works directly on the encrypted version of the data. Our model consists of an encrypted database at an *untrusted* central site. Queries are coded for the encrypted data. Most query processing occurs at the central site with a final interpretation occurring at a secure local workstation, along with query initiation and coding.

We do not know methods which would provide the functionality of a modern DBMS within this model. However, we believe there are specialized applications where security can be enhanced without affecting the application.

[1] Research supported in part by NSF grant DCR-8403350 and by a Research Scholar award from Drexel University.

[2] On leave 1984-86, Drexel University.

It is useful to identify attacks against an encrypted database. In a *static attack*, an opponent acquires a copy of the database as it exists when not in use. (With an unencrypted database, the opponent will then have succeeded completely. With an encrypted database, this attack might yield information even if the opponent is not able to decrypt.) In an *offline* static attack, an opponent gets a single copy of the database, while in an *online* static attack, an opponent gets any number of copies, including copies that differ only by one or a few transactions.

With a *dynamic attack*, an opponent can monitor the processing of the database at the central site, observing queries and the resulting processing. We have the *known query* version, where opponents also know the original uncoded form of queries, and the *chosen query* version, where opponents can originate arbitrary queries.

Section 2 below gives an overview of security approaches. Section 3 discusses a specialized method that uses encrypted blocks with a separate concordance of pointers into these blocks. For this specialized application, the method seems to provide good security even against dynamic attacks. Section 4 discusses a more general approach based on homophonic ciphers. This approach seems to give good security against static attacks, though the security is somewhat data dependent. It also gives moderate security against dynamic attacks.

2. Overview of approaches to database security

In attempting to enhance the security of a database, researchers have considered a number of different approaches. We give brief descriptions of some of these approaches ((a) through (h) below), followed by brief references to the two specialized approaches considered in this paper ((i) and (j)).

(a) *Trusted kernel.* Here one isolates a security kernel in the DBMS software, and proves that the kernel can enforce security and that it is correct. This is a very promising approach that is being actively pursued. It does not involve encryption at all. Notice that the software which has been proved correct must be compiled with a secure compiler and must run in unaltered form on a secure operating system [Tho84].

(b) *Trusted filter.* Here a simple trusted filter is placed between an untrusted DBMS and its users. The method uses cryptographic checksums to insure that data does not directly leak out to users not cleared for the data [Den84a]

[Gra84] [AFS83]. This approach provides a reasonable short-term solution to security problems. [Den85] describes an enhancement to this scheme.

(c) *Restricted views.* One arranges the database so that a user cleared to a certain level gets a view that is missing all data at higher levels. This could be arranged in a system like MULTICS by storing data cleared for different levels in different security rings. As in approach (d) below, one could also use different encryption keys for different levels.

(d) *Record or block encryption -- single key.* This is a straight-forward method that easily extends to multi-level security with the use of multiple keys. It is also an archival method. Processing of the database requires large amounts of decryption.

(e) *Record encryption -- separate field keys.* This is an interesting approach that combines record encryption with separate field decryption. Only a single very specialized implementing cryptosystem has been suggested [Dav81] [Dav82] [Oma83]. The method seems inferior to (g) in most respects [Den84b]. See also [Bla85a] for a similar approach.

(f) *Field encryption -- single key.* This approach is vulnerable to ciphertext searching / substitution. We get this same weakness as long as a single key is used for the same field within different records.

(g) *Field encryption -- separate keys for each field of each record.* Rather than store a key for each field of each record, the keys can be generated as needed as a one-way function of a record id, a field id, and a single secret key. This approach is attractive in many ways, but requires quite a bit of decryption and key generation [Den84b].

(h) *Privacy homomorphisms.* This elegant idea would allow direct manipulation of the data without decryption. There are theoretical limitations to the method, and the only implementing cryptosystems are specialized ones with possible hidden weaknesses [Riv78]. As a form of single-key field encryption, this is also vulnerable to ciphertext searching / substitution. [Fei85] presents an extension of this idea to more general problems.

(i) *Record or block encryption -- coded files of pointers to blocks.* In this case, one locates a desired block by looking up a coded entry and finding the block number. This method seems useful for certain specialized applications. More details are presented in Section 3.

(j) *Homophonic representation of data in subfields.* Briefly stated, this approach is a combination of two ideas:

- Break fields into subfields to facilitate processing the database in encrypted form using counting techniques. (This gives the advantages of the privacy homomorphism approach.)

- Create multiple (homophonic) representations of data in a subfield to flatten the distribution of data in each subfield. (This will alleviate the weaknesses associated with field encryption.)

See Section 4 for details.

3. Encrypted blocks with concordance

In this section we describe an encryption scheme suitable for keyword-based searches of a large text file. By "large" we mean 100 megabytes or larger -- too large to be kept at a local workstation. [Bla85b] gives an example of such a large database, managed by the STAIRS full-text retrieval system.

We wish to store the file at an untrusted central site in encrypted form, yet retain as much functionality as we can. The method we propose relies on a *concordance* to do searches (at the central site) of the file. In its simplest form, this scheme breaks the file into encrypted blocks. As we encrypt the file, we construct a concordance, which lists all words which occur in the file, and for each such word a list of all blocks in which the word occurs. The words are listed in the concordance using numerical codes, and the concordance (which is large) is stored at the central site. (All the functions provided by STAIRS can be implemented in the encrypted environment we describe here.)

To encode the words, we select in advance a true random permutation of the integers between 1 and 10000 (say) and use this permutation to assign numerical codes to the words in the order of occurrence in the text.

A user needs the key to decrypt blocks and a list of the words and their corresponding codes. Queries take the form of *keyword searches*, i.e., the user sends the central site a set of keyword codes, and the central site returns every block which contains all of the requested keywords. The user then decrypts these blocks.

Suppose the text file is 100 megabytes long and we have 10K distinct words in the file. If each block is 1K long, then we have 100,000 blocks, so we need at least 17 bits to store each block number. Each word occurs, on the average, in 1,000 blocks (assuming an average wordsize of 10 bytes) so we need approximately 20 megabytes for our concordance in its simplest form. If the encoding of the keywords is kept as a (sorted) table, this requires only about 120K bytes -- well within the capabilities of a typical workstation.

A number of advantages of the scheme can be listed.

- The file resides in encrypted form and is thus secure against a static attack. In particular, an opponent who gains unauthorized access to the central site's disk hardware would not be able to profit from a wholesale dump of the disk contents.

- Assuming the kind of searching described is all we need to do, there is very little overhead imposed by the encryption of the file, since we would probably want a concordance similar in size and form to the one we have constructed even if the file were stored in the clear. Essentially no extra processing time or storage is required.

- Finding and retrieving the proper blocks can be carried out at the untrusted central site. The central authority does not need to know the encryption key or the codes for the words in the file.

- The scheme lends itself easily to multi-level security classification: we simply encrypt different parts of the file with different keys depending on the classification level.

- Archiving the file is trivial since no further encryption is required.

- The database along with the concordance can be created incrementally, where the local site supplies information to update the concordance as it adds the new blocks to the encrypted file. An eavesdropper gets the information that some new words have been added to the file (in the case of words which have not previously occurred), and the codes for these new words are known, although not the actual words. The incremental approach has the advantage that the textfile is never stored or transmitted in the clear, so the central site need not be secure.

One can think of several ways of enhancing the scheme:

- The user can embed information in the file to make searches more effective. For example the block numbers of related material could be embedded as special words, e.g. 1$ 0$$ 0$$$ 4$$$$ 2$$$$$ might be an encoding for block number 10,042. Note that the concordance would not be able to store numbers as words, since that might make it too large. It would also be easy to store words from document titles or index terms in a special form.

- To cut down the number of "false alarms" resulting from a search where the "keyword set" is used to look for a phrase, the concordance could store, in addition to the block number, the code of the following word.

- If the chosen limit on the number of entries in the concordance (10,000 in our example) causes difficulty, one might allow users to remove items from the concordance and substitute new words. The central site would simply remove the blocklists for certain codes and substitute others.

Finally, we suggest several randomization methods for adding protection against statistical attacks. However, it is not clear that these methods change the security of the system in any significant way. Note that an opponent who has access to the concordance knows only that some (unknown) words occur in many blocks, while others occur in few blocks.

- Treat some of the most common words as "noise" words -- they would not appear in the concordance.

- Use multiple codes for words with long block lists. This would enable these lists to be broken into smaller (not necessarily disjoint) sublists. These new codes are introduced as needed during the incremental creation of the concordance.

- Add dummy blocks.

- Add bogus block numbers (referencing legitimate blocks) to certain block lists.

In the previous two items, the local workstation can keep track of dummy blocks and bogus references, provided these are not too numerous, or they can be filtered out along with the false alarms mentioned earlier.

It can be argued that as the power of a personal workstation grows it will be increasingly feasible to keep a copy of the entire database at each workstation. This will be true for many applications, but there are other areas where it is unacceptable to make a separate copy for each user. In particular, with our system the central site can keep a record of the parts of the database each user accesses. (One would have to trust the central site to do this data logging, though it would still not be able to decrypt.)

A database similar to the one described in this section has apparently been implemented as part of a classified project.

4. The homophonic encrypted database

Here we use field encryption and counting techniques to allow the kinds of processing permitted by privacy homomorphisms. For example, if some statistical property of the database is needed, the central site will prepare a *distribution vector* of the encrypted values for interpretation at a secure local workstation. Such vectors would often be too large, so it is necessary to break fields into small subfields. In order to prevent ciphertext searching, we use multiple representations of data values in a given subfield. This adds "noise" to the subfield value, though we prefer to regard the approach more formally as a homophonic cipher, i.e. an encryption scheme with multiple possible cipher codes for a given plaintext symbol [Den82]. (One of the codes is chosen at random.) The homophonic approach also takes care of ciphertext substitution, though this problem could easily be handled with a secure encrypted checksum ("spraypaint").

4.1. The homophonic encoding

For simplicity of discussion, we first assume the database consists of r fixed-size records of text. (Later we discuss extensions to the relational model.) Assume each record is broken into fixed-length fields and each field into a sequence of characters, so that the entire record can be regarded as a fixed-length character string.

We will handle each character position in the records separately. If we focus on a single position, we have r characters to work with, one from each record. We will show how to use a homophonic encoding of these r characters so that the distribution after encoding is exactly flat (i.e. the same number of occurrences of each code). Assuming the order of records is randomized, an opponent would get no information from any single character position. Of course statistics on pairs (or n-tuples, $n > 1$) of coded values might yield some information.

In order to achieve a perfectly flat distribution, we must add extra dummy records. The number of dummy records needed will depend upon parameters of the homophonic encoding, but will not be too large for reasonable parameters. To make the discussion simpler, assume each character has a 6-bit representation, and assume we expand to an 8-bit homophonic encoding. (The arguments below work just as well for any other pair besides 6 and 8.) For these parameters, we will need approximately $r/3$ dummy characters to be certain that a homophonic coding exists with a flat distribution. More precisely, we have the following result.

Theorem. Suppose we have r 6-bit characters and wish to use an 8-bit homophonic encoding. Choose an integer m as small as possible so that $r \leq 192*m + 64$. Add extra dummy characters so that there are $256*m$ altogether. It is always possible to choose characters for the dummy positions and to choose 8-bit homophonic representations for all characters so that there are *exactly* m occurrences of each of the 256 8-bit values. There are examples where no fewer number of extra characters will work.

We will only outline the proof here. Create 256 "bins", each holding m characters. Allocate characters to bins from the r initial characters, putting only the same character value in a given bin and not starting a new bin for a given character until the old bin is full. In this way, we never have two bins partly full of the same character. Thus we can have at most 64 partly full bins. Assuming r satisfies the conditions of the theorem, we can show that we never need more than 256 bins. Each bin corresponds to an 8-bit homophonic representation.

In order to complete the encoding of the entire file, we encode each character position separately as above. Then randomize the order of the character positions, the same way for each record. Finally randomize the order of the records. Assuming there are n character positions in a record, the secret key consists of a random permutation of n items, and n separate homophonic ciphers, each mapping 64 6-bit characters *onto* 256 8-bit values. (For query processing, we may want each possible character included at least in the dummy records, if nowhere else.) The key can easily be encoded in $n * \log n + 256*6*n$ bits or approximately $200*n$ bytes.

The static appearance of this database, assuming no additional queries have been processed, is like a $256*m$ by n array of 8-bit values. Taking a slice with the second coordinate fixed, we get perfectly "flat" statistics: exactly m occurrences of each of the possible 256 8-bit values. The difficulty of cryptanalysis of a single static copy, even if the contents were known, would depend on the degree to which data items are correlated with one another. An uncorrelated item would be unconditionally secure, and analysis seems very difficult even for highly correlated data. (A given field is spread across a secret random set of character positions, and each position has many representations for each character.)

In practice we could usually get by with fewer dummy records and still achieve exactly flat distributions in each character position. Besides, an *exactly* flat distribution does not seem to contribute much extra security. Without extra effort, the perfect distribution would be lost after any update, and such flat distributions would not be possible for positions in fields common to several relations in a relational database. (See Section 4.3.)

4.2. Query processing

Assuming the database resides at an untrusted central site, we can still get most of the functions of a modern DBMS while allowing most of the processing to occur at the central site. Initial query coding and final query interpretation must occur at a secure local workstation.

For example, we can retrieve an arbitrary record based on the value of some key field. One just needs to specify the different homophonic representations for each character in the key -- at most 256 8-bit numbers.

More generally, we can start with an arbitrary boolean expression of the possible entries in various fields. The central site will be able to retrieve the (encrypted) records corresponding to the expression by using lists of possibilities for each character position. For instance, suppose we want to retrieve records satisfying $18000 \leq \text{salary} \leq 31999$. Suppose further that the salary field is represented using 5 characters, and that the first two characters are stored in positions j and k in the encrypted file. The local workstation makes up a query like:

((position j represents 1) **and**
((position k represents 8) **or**
(position k represents 9))) **or**
(position j represents 2) **or**
((position j represents 3) **and**
((position k represents 0) **or**
(position k represents 1)))

Each part of the query of the form "position j represents 2" is coded as a list of at most 256 8-bit homophonic values. Thus this query requires at most $256*7 \approx 2K$ bytes.

Notice that an opponent gets extra information about the homophonic encoding from eavesdropping on such a query.

As another example of possible queries, one can calculate *exact* averages over values in some numeric field. Such averages can be calculated for a subset of the records specified by a boolean formula as above. For each character position in the numeric field and for each record specified by the formula the local workstation asks the central site to compute a *distribution vector* giving a count of the number of occurrences of each of the 256 homophonic values. The local workstation knows the homophonic encoding, so it can properly interpret the vector. Just adding up averages over each character position gives an exact average over the entire field.

Notice that it is necessary to eliminate any dummy records in such a calculation. For this purpose we need two extra character positions in each record identifying the dummy records. (With only one position, we can not guarantee a flat homophonic encoding.) This means that a dynamic attack can identify and eliminate the dummy records.

4.3. Enhancements

We can use this approach on a more general database model than a file of records. In particular we want relational joins. For such joins to work, the homophonic representations must match up for the same fields in different relations. This means we must give up an exactly flat distribution. More importantly though, we must tell the central site which homophones represent the same character for those character positions in fields over which the join is being taken. While not directly allowing decryption, this certainly gives information to a dynamic attack.

Exact variances (and exact standard deviations) can be calculated over arbitrary subsets of some numeric field. For this purpose we would store (in homophonic encrypted form) the *square* of the field value as well as the field value itself. Then an exact variance calculation is similar to the calculation of an exact average mentioned in Section 4.2.

There are a number of other enhancements to improve security against dynamic attacks (those with access to the coded queries).

- Batch queries and updates together.
- When inserting a new record, replace a dummy record with the new one. (This makes it harder to distinguish between an update and an insertion.)
- When making an update of a field, make random changes to other character positions. (This is possible because of the homophonic encoding.)
- Periodically redo completely the encoding of the entire database.
- Add extra dummy queries, and dummy parts to real queries.
- Add extra dummy character positions.

The system parameters of 64 (= number of characters) and 256 (= number of homophonic codes) were chosen for illustrative purposes and to balance competing factors. A larger number of homophonic codes leads to greater security and to a smaller number of dummy records required to achieve flat distribution. But increasing the number of homophonic codes also leads to file expansion, greater processing time and increased key size. (There is a portion of the key for each character position. For a fixed position, that portion must give the character corresponding to each homophonic code.) The amount of traffic between the local workstation and the central site is roughly independent of the number of homophonic codes, since this traffic mainly consists of distribution vectors (= counts of the number of occurrences) of homophonic codes, and if there are more codes, the counts for each code are correspondingly smaller.

For example, if we specified 1024 homophonic codes, rather than 256, the database would stay about the same size (longer homophonic code, but fewer dummy records), and the amount of traffic would also stay about the same while the key size would increase by a factor of 4 (not counting the part of the key that gives a permutation of character positions). The processing time would also increase at both the local workstation and the central site. An extreme and apparently impractical special case occurs with as many homophonic codes as records and with no dummy records.

5. Conclusions

For certain specialized applications, we can incorporate encryption into a database with little penalty or loss of function. Most work is carried out directly on the encrypted data. Encryption gives the database an extra degree of security against theft, browsing and hardware failures, and allows storage at an insecure facility.

Acknowledgement

The first author would like to thank Roger Bradford and Dorothy Denning for conversations about the systems described here.

References

[AFS83] Air Force Studies Board, "Multilevel Data Management Security," National Research Council, National Academy Press, Washington, D. C., 1983.

[Bla85a] G. R. Blakley and C. Meadows, "A database encryption scheme which allows the computation of statistics using encrypted data," *Proceedings of the 1985 Symposium on Security and Privacy*, IEEE Computer Society, pp. 116-122.

[Bla85b] D. C. Blair and M. E. Maron, "An evaluation of retrieval effectiveness for a full-text document-retrieval system," *Communications of the ACM* 28, 3 (March 1985), pp. 289-299.

[Dav81] G. I. Davida, D. L. Wells and J. B. Kam, "A database encryption system with subkeys," *ACM Transactions on Database Systems*, 6, 2 (June 1981), pp. 312-328.

[Dav82] G. I. Davida and Y. Yeh, "Cryptographic relational algebra," *Proceedings of the 1982 Symposium on Security and Privacy*, IEEE Computer Society, pp. 111-116.

[Den82] D. E. Denning, *Cryptography and Data Security*, Addison-Wesley, 1982.

[Den84a] D. E. Denning, "Cryptographic checksums for multilevel database security," *Proceedings of the 1984 Symposium on Security and Privacy*, IEEE Computer Society, 1984, pp. 52-61.

[Den84b] D. E. Denning, "Field encryption and authentication," *Advances in Cryptology: Proceedings of Crypto 83*, ed. by D. Chaum, Plenum, 1984, pp. 231-247.

[Den85] D. E. Denning, "Commutative filters for reducing inference threats in multilevel database systems," *Proceeding of the 1985 Symposium on Security and Privacy*, IEEE Computer Society, pp. 134-146.

[Fei85] J. Feigenbaum, "Encrypting problems instances, or ..., how to take advantage of someone without having to trust him," presentation at Crypto 85.

[Gra84] R. Graubart, "The integrity-lock approach to secure database management," *Proceedings of the 1983 Symposium on Security and Privacy*, IEEE Computer Society, 1984, pp. 62-74.

[Oma83] K. A. Omar and D. L. Wells, "Modified architecture for the sub-keys model," *Proceedings of the 1983 Symposium on Security and Privacy*, IEEE Computer Society, 1984, pp. 79-86.

[Riv78] R. L. Rivest, L. Adelman and M. L. Dertouzos, "On data banks and privacy homomorphisms," *Foundations of Secure Computation*, ed. by R. A. DeMillo et al., Academic Press, 1978, pp. 169-179.

[Tho84] K. Thompson, "Reflections on Trusting Trust," *Communications of the ACM* 27, 8 (Aug. 1984), pp.761-763.

[Wag81] N. R. Wagner, "Practical approaches to secure computer systems," *Technical Report UH-CS-81-3*, Computer Science Department, University of Houston, Texas, April 1981.